Lecture 1

Useful Standard Python Libraries

Python has a number of built-in libraries, or collections of modules, that provide a wide range of functionality. Some of the most commonly used built-in libraries include:

- os: provides a way to interact with the operating system
- sys: provides access to system-specific parameters and functions
- math: provides mathematical functions and constants
- random: generates random numbers
- string: provides string-related functions
- re: provides regular expression matching operations
- json: provides encoding and decoding of JSON data
- datetime: provides classes for manipulating dates and times
- collections: provides alternatives to built-in types that can be more efficient in certain situations

Let's take a look at the Python Standard Library, which is a set of modules included with every Python installation. These modules provide a wide range of functionality and are designed to improve efficiency and ease of use. You learned about the first modules in Python classes, so now we will look at other modules.

collections: This library provides alternatives to built-in types that can be more efficient in certain situations. Some examples of classes in this library include Counter, which is a dict subclass for counting hashable objects, and defaultdict, which is a dict subclass that calls a factory function to supply missing values.

Counter: This class is a dict subclass for counting hashable objects. It is a container that will hold the count of each of the elements present in the container.

Example:

```
from collections import Counter
c = Counter()
for word in ['red', 'blue', 'red', 'green', 'blue', 'blue']:
    c[word] += 1
print(c)
# Output: Counter({'blue': 3, 'red': 2, 'green': 1})
```

defaultdict: this class is a dict subclass that calls a factory function to supply missing values. It is similar to the standard dictionary, but it has a default value for any new keys that are accessed.

Example:

```
from collections import defaultdict
d = defaultdict(int)
d['a'] = 1
d['b'] = 2
print(d['c']) # 0
```

itertools: this module provides a variety of functions for working with iterators, including functions for creating iterators, combining them, and working with infinite iterators. For example, itertools.count() returns an iterator that generates an infinite sequence of integers, starting with a given value, and itertools.product() returns the cartesian product of input iterables as an iterator.

cycle: this function from itertools returns an iterator that will cycle through the elements of an iterable indefinitely.

Example:

```
from itertools import cycle
colors = cycle(['red', 'blue', 'green'])
for i in range(7):
    print(next(colors))
# Output: red blue green red blue green red
```

repeat: this function from itertools returns an iterator that will repeat the same element a specified number of times or indefinitely if no number is specified

Example:

```
from itertools import repeat
for i in repeat('over', 5):
    print(i)
# Output: over over over over over
```

All these libraries are very useful and can be used in various situations. They provide functionality that is often needed in programming, and they are well-tested and optimized to be efficient.

re: The re module provides regular expression matching operations. Regular expressions are a powerful tool for matching patterns in text, and the re module provides a wide range of functions for working with regular expressions in Python. Some common functions in this module include search(), findall(), sub(), and compile().

Example:

python

Example:

```
import re

# Search for the first occurrence of a pattern
result = re.search(r'\d{3}-\d{2}-\d{4}', 'Social Security Number: 123-45-6789')
print(result.group(0))  # 123-45-6789

# Find all occurrences of a pattern
result = re.findall(r'\b\w+\b', 'This is a sentence with a few words')
print(result)  # ['This', 'is', 'a', 'sentence', 'with', 'a', 'few', 'words']

# Replace all occurrences of a pattern with a new string
result = re.sub(r'\d{3}-\d{2}-\d{4}', 'xxx-xx-xxxx', 'Social Security Number: 123-45-6789')
print(result)  # Social Security Number: xxx-xx-xxxx
```

json: The json module provides encoding and decoding of JSON (JavaScript Object Notation) data. JSON is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. The json module provides functions for working with JSON data in Python, including load(), loads(), dump(), and dumps().

Example:

```
import json

# Serialize a Python object to a JSON formatted string
data = {'name': 'John', 'age': 30, 'city': 'New York'}
json_data = json.dumps(data)
print(json_data)  # {"name": "John", "age": 30, "city": "New York"}

# Deserialize a JSON string to a Python object
json_data = '{"name": "John", "age": 30, "city": "New York"}'
data = json.loads(json_data)
print(data)  # {'name': 'John', 'age': 30, 'city': 'New York'}
```

string: The string module provides string-related functions. It contains a number of useful string constants, such as ascii_letters, digits, and punctuation, as well as functions for working with strings, such as capwords(), join(), replace(), and split().

Example:

```
import string

# Capitalize the first letter of each word in a string
result = string.capwords('this is a test')
print(result)  # This Is A Test

# Concatenate a list of strings with a separator
words = ['hello', 'world']
```

All these libraries are very useful and can be used in various situations. They provide functionality that is often needed in programming, and they are well-tested and optimized to be efficient.

JSON and CSV

JSON (JavaScript Object Notation) and CSV (Comma-Separated Values) are two commonly used data formats for exchanging and storing data.

JSON is a human-readable, text-based format that is commonly used for data exchange in APIs and web applications. It uses key-value pairs to represent data and is often used for storing complex data structures, such as nested arrays and dictionaries. JSON is also easy to parse and generate using many programming languages, including Python.

CSV, on the other hand, is a simpler and more widely supported format. It stores data in a tabular form, where each line is a record and each field within a record is separated by a comma. This format is widely used in data analysis and spreadsheet applications, and is supported by most database and spreadsheet programs. The simplicity of CSV makes it easy to use and transfer data, but it is not well suited for representing complex data structures.

In summary, JSON is a more versatile and flexible data format, while CSV is simpler and more widely supported. The choice between JSON and CSV depends on the specific needs of the data being stored and the applications that will be accessing it.

Examples of when you might use JSON or CSV:

- **JSON: When you want to exchange data between a web application and a server**, you might use JSON. For example, you might use an API to fetch data from a server and receive the data in a JSON format. You can then use the JSON data in your web application.
- **CSV: When you want to analyze data in a spreadsheet program**, you might use CSV. For example, you might export data from a database and save it as a CSV file, which you can then open in a spreadsheet program like Microsoft Excel or Google Sheets. You can easily sort, filter, and visualize the data in the spreadsheet program.
- **JSON: When you want to store complex data structures**, you might use JSON. For example, you might use JSON to store information about a collection of books, where each book has a title, author, publication date, and list of genres. You can easily nest this data within the JSON format to represent the relationships between the different elements.

- **CSV: When you want to store data that is easy to process**, you might use CSV. For example, you might use CSV to store a list of names and addresses, where each line is a record and each field within a record is separated by a comma. You can easily manipulate this data using simple string operations and write it to a file for later use.

Examples comparing JSON and CSV formats in Python:

JSON:

```
import json

# A Python dictionary to be converted to JSON
person = {
  "first_name": "John",
  "last_name": "Doe",
  "age": 30,
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA"
  }
}

# Convert the dictionary to JSON string
person_json = json.dumps(person)

# Load the JSON string to a Python dictionary
person_dict = json.loads(person_json)

print("Person JSON:", person_json)
print("Person Dict:", person_dict)
```

CSV:

```
import csv

# A list of dictionaries to be converted to CSV
people = [
  {
    "first_name": "John",
    "last_name": "Doe",
    "age": 30,
    "address": "123 Main St, Anytown, CA"
  },
  {
    "first_name": "Jane",
    "last_name": "Doe",
```

```
    "age": 28,
    "address": "456 Elm St, Anytown, CA"
  }
]

# Write the list of dictionaries to a CSV file
with open('people.csv', 'w', newline='') as csvfile:
    fieldnames = ['first_name', 'last_name', 'age', 'address']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    for person in people:
        writer.writerow(person)

# Read the CSV file to a list of dictionaries
with open('people.csv', 'r') as csvfile:
    reader = csv.DictReader(csvfile)
    people_list = list(reader)

print("People List:", people_list)
```

В приведенных выше примерах JSON используется для представления одного словаря, а CSV — для представления списка словарей. Преобразование между двумя форматами можно выполнить с помощью библиотек json и csv.

In the examples above, JSON is used to represent a single dictionary while CSV is used to represent a list of dictionaries. The conversion between the two formats can be done using the json and csv libraries.

Practical tasks on the topic of this lecture

Processing JSON data: Using the json module, write a script that reads in a JSON file containing information about books (title, author, and ISBN) and outputs a list of all the books written by a specific author.

Example:

```
import json

def find_books_by_author(file_path, author):
    with open(file_path, 'r') as f:
        data = json.load(f)
    books = []
```

```
    for book in data['books']:
        if book['author'] == author:
            books.append(book)
    return books

file_path = 'books.json'
author = 'J.K. Rowling'
books = find_books_by_author(file_path, author)
print(books)
```

Cleaning text data: Using the string module, write a script that takes in a text file and removes all punctuation and capitalization, and then write the cleaned text to a new file.
python
Example:

```
import string

def clean_text(file_path):
    with open(file_path, 'r') as f:
        text = f.read()
    text = text.lower()
    text = text.translate(text.maketrans('', '', string.punctuation))
    with open('cleaned_text.txt', 'w') as f:
        f.write(text)

file_path = 'text.txt'
clean_text(file_path)
```

Manipulating CSV data: Using the csv module, write a script that reads

Example:

```
import csv

def calculate_avg_income(file_path):
    with open(file_path, 'r') as f:
        reader = csv.DictReader(f)
        income_by_state = {}
        for row in reader:
            state = row['state']
            income = int(row['income'])
            if state in income_by_state:
                income_by_state[state]['total'] += income
                income_by_state[state]['count'] += 1
```

```
            else:
                income_by_state[state] = {'total': income, 'count': 1}
    avg_income_by_state = {state: income_by_state[state]['total'] / income_by_state[state]['count'] for state in income_by_state}
    return avg_income_by_state

file_path = 'income_data.csv'
avg_income = calculate_avg_income(file_path)
print(avg_income)
```

Extracting information from web pages: Using the re module, write a script that extracts all the phone numbers from a web page.

python

Example:

```
import re
import requests

def extract_phone_numbers(url):
    r = requests.get(url)
    phone_numbers = re.findall(r'\(?\d{3}\)?[-.\s]?\d{3}[-.\s]?\d{4}', r.text)
    return phone_numbers

url = 'https://example.com'
phone_numbers = extract_phone_numbers(url)
print(phone_numbers)
```

These are just a few examples of practical tasks that can be given using these modules. The specific tasks will depend on the skill level of the students and the goals of the course. It's also worth mentioning that these examples are simplified versions of what one might encounter in a real-world scenario.

Counting occurrences of items in a list: Using the Counter class, write a script that takes in a list of items and returns a dictionary of the items and their occurrences in the list.

Example:

```
from collections import Counter

def count_items(items):
    return Counter(items)

items = [1, 2, 3, 4, 1, 2, 3, 1, 2, 1]
counts = count_items(items)
print(counts)
```

Accessing missing dictionary keys: Using the defaultdict class, write a script that takes in a list of items and a default value, and returns a dictionary of the items and their occurrences in the list. If an item is not in the dictionary, it should return the default value.

Example:

```
from collections import defaultdict

def count_items(items, default_value):
    counts = defaultdict(lambda: default_value)
    for item in items:
        counts[item] += 1
    return counts

items = [1, 2, 3, 4, 1, 2, 3, 1, 2, 1]
default_value = 0
counts = count_items(items, default_value)
print(counts)
```

Iterating over combinations: Using the itertools module, write a script that takes in a list of items and a number n, and returns all possible combinations of n items from the list.

Example:

```
import itertools

def get_combinations(items, n):
    return list(itertools.combinations(items, n))

items = [1, 2, 3, 4]
n = 2
combinations = get_combinations(items, n)
print(combinations)
```

Repeating an iterator: Using the itertools module, write a script that takes in an iterator and a number n, and returns the iterator repeated n times.

Example:

```
import itertools

def repeat_iterator(iterable, n):
    return itertools.chain.from_iterable(itertools.repeat(iterable, n))

iterable = [1, 2, 3]
```

```
n = 2
repeated_iterable = repeat_iterator(iterable, n)
print(list(repeated_iterable))
```

Cycling through an iterator: Using the itertools module, write a script that takes in an iterator and returns an infinite iterator that cycles through the input iterator.
Example:

```
import itertools

def cycle_iterator(iterable):
    return itertools.cycle(iterable)

iterable = [1, 2, 3]
cycled_iterable = cycle_iterator(iterable)
print(next(cycled_iterable))
print(next(cycled_iterable))
print(next(cycled_iterable))
print(next(cycled_iterable))
```

Lecture 2

Efficient Data Structures

pandas and NumPy are two popular Python libraries that are widely used in data analysis and manipulation.

Pandas is a library for data manipulation and analysis. It provides data structures and functions for handling and manipulating numerical tables and time series data. It is built on top of NumPy and provides easy-to-use data structures and data analysis tools for handling and manipulating numerical tables and time series data.

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Grouping and aggregation: pandas provides powerful tools for grouping and aggregating data. You can group data by one or more columns and apply various aggregation functions to each group. For example, you can group a dataset by the values in a column and then calculate the mean of each group.

Example:

```
import pandas as pd
df = pd.read_csv('data.csv')
grouped_data = df.groupby('column_name').mean()
print(grouped_data)
```

Handling missing data: pandas provides various options for handling missing data, such as filling in missing values with a specific value or interpolating the missing values.

Example:

```
import pandas as pd
df = pd.read_csv('data.csv')
df = df.fillna(value=0)
print(df)
```

Merging and joining data: pandas provides various options for merging and joining data from multiple DataFrames or datasets. For example, you can join two DataFrames on a specific column, similar to a SQL JOIN.

Example:

```
import pandas as pd
df1 = pd.read_csv('data1.csv')
df2 = pd.read_csv('data2.csv')
```

```
merged_data = pd.merge(df1, df2, on='column_name')
print(merged_data)
```

Data transformation: pandas provides various functions for transforming data, such as pivoting and melting data. For example, you can pivot a DataFrame to reshape it into a different format.

Example:

```
import pandas as pd
df = pd.read_csv('data.csv')
pivoted_data = df.pivot(index='column_name1', columns='column_name2', values='column_name3')
print(pivoted_data)
```

pivot is a DataFrame method in the pandas library that can be used to reshape a DataFrame. It allows you to transform a DataFrame from a "long" format to a "wide" format or vice versa.

The basic syntax for using the pivot method is as follows:

```
pivot_table = df.pivot(index='index_column', columns='column_to_become_columns', values='column_to_become_values')
```

Here, index_column is the column that will become the new DataFrame's index, column_to_become_columns is the column that will become the new DataFrame's columns, and column_to_become_values is the column that will populate the new DataFrame's cells.

For example, let's say you have a DataFrame with three columns: 'date', 'product', and 'sales'. To pivot this DataFrame so that the 'date' column becomes the index, the 'product' column becomes the columns, and the 'sales' column becomes the values, you would use the following code:

```
import pandas as pd
df = pd.read_csv('data.csv')
pivoted_data = df.pivot(index='date', columns='product', values='sales')
print(pivoted_data)
```

Another useful feature of the pivot function is the ability to pass an optional argument aggfunc to it. This allows you to specify the aggregation function that is applied to duplicate values. By default, pivot will raise an error if there are duplicate values.

For example, if you have the following DataFrame:

```
  Product  Sales  Date
0   Apple    100    1
```

1 Apple 200 1
2 Pear 50 2
3 Pear 30 2

You can use the following code to pivot this dataframe and sum the duplicate values:

```
import pandas as pd
df = pd.read_csv('data.csv')
pivoted_data = df.pivot_table(index='Date', columns='Product', values='Sales',aggfunc='sum')
print(pivoted_data)
```

The resulting pivot table would be:

```
Product  Apple  Pear
Date
1          300   NaN
2           NaN   80
```

Note that pivot_table is similar to pivot but will handle duplicate values by applying the specified aggregation function.

pivot and pivot_table are powerful tools for reshaping and aggregating data in pandas. These methods can be used to transform a DataFrame into a format that is more suitable for a specific task or analysis.

Data visualization: pandas integrates well with data visualization libraries such as matplotlib and seaborn, which allows you to easily create plots and charts from your DataFrames.

Example:

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df.plot(kind='bar',x='column_name1',y='column_name2')
plt.show()
```

This is just a small sample of the capabilities of pandas. It's a very versatile library and can be used to perform a wide range of data manipulation and analysis tasks.

matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

It provides a range of tools for creating static, animated, and interactive visualizations in Python. Some of the most common types of plots that can be created with matplotlib include:

- Line plots: for displaying data that changes over time or other continuous variables
- Scatter plots: for displaying the relationship between two or more variables
- Bar plots: for displaying data that is divided into different categories
- Histograms: for displaying the distribution of a single variable
- Pie charts: for displaying the proportion of different categories in a dataset

Example of how to use matplotlib to create a simple line plot:

```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a figure and axis
fig, ax = plt.subplots()

# Plot the data
ax.plot(x, y)

# Add labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Simple Line Plot')

# Show the plot
plt.show()
```

matplotlib also has an extensive customization options, you can easily change the color, line style, and marker style of your plots. You can also add gridlines, legends, and annotations to your plots. Also, you can use a variety of pre-defined styles and color palettes, such as "ggplot" or "seaborn".

There are other library like seaborn that built on top of matplotlib, it provides a high-level interface for drawing attractive and informative statistical graphics.

For more advanced usage, matplotlib also has capabilities for creating 3D plots, subplots, and other advanced visualizations. With a little bit of creativity and knowledge of the library's capabilities, you can create all sorts of interesting and informative visualizations with matplotlib.

For practical tasks, you can use matplotlib to visualize data from a CSV file, visualize the distribution of a dataset, or compare multiple datasets in a single plot. You can also use it to create plots of mathematical functions or simulate simple animations.

Visualizing data from a CSV file:

```
import matplotlib.pyplot as plt
import pandas as pd

# Read data from a CSV file
data = pd.read_csv('data.csv')

# Create a scatter plot of the data
plt.scatter(data['x'], data['y'])
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot of Data')
plt.show()
```

Visualizing the distribution of a dataset:

```
import matplotlib.pyplot as plt
import numpy as np

# Create a random dataset
data = np.random.normal(50, 10, 100)

# Create a histogram of the data
plt.hist(data, bins=20)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of Data')
plt.show()
```

Comparing multiple datasets in a single plot:

```
import matplotlib.pyplot as plt
import numpy as np

# Create two random datasets
data1 = np.random.normal(50, 10, 100)
data2 = np.random.normal(60, 15, 100)
```

```
# Create a figure and axis
fig, ax = plt.subplots()

# Plot both datasets on the same axis
ax.plot(data1, label='Dataset 1')
ax.plot(data2, label='Dataset 2')

# Add labels, title, and legend
ax.set_xlabel('Index')
ax.set_ylabel('Value')
ax.set_title('Comparison of Two Datasets')
ax.legend()

# Show the plot
plt.show()
```

Plotting a mathematical function:

```
import matplotlib.pyplot as plt
import numpy as np

# Create x and y data
x = np.linspace(-np.pi, np.pi, 100)
y = np.sin(x)

# Create a figure and axis
fig, ax = plt.subplots()

# Plot the data
ax.plot(x, y)

# Add labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Plot of the Sine Function')

# Show the plot
plt.show()
```

Simulating a simple animation:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
# Create x and y data
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

# Create a figure and axis
fig, ax = plt.subplots()

# Initialize a line object
line, = ax.plot(x, y)

# Define an animation function
def animate(i):
    line.set_ydata(np.sin(x + i / 10))
    return line,

# Create an animation object
ani = animation.FuncAnimation(fig, animate, frames=100, blit=True)

# Show the animation
plt.show()
```

You can find more information and examples on the official website of matplotlib: https://matplotlib.org/stable/contents.html

Practical tasks on the topic of this lecture

Loading a CSV file into a DataFrame: Using pandas, write a script that loads a CSV file into a DataFrame and displays the first 5 rows of the DataFrame.

Example:

```
import pandas as pd

df = pd.read_csv('data.csv')
print(df.head())
```

Selecting columns from a DataFrame: Using pandas, write a script that selects specific columns from a DataFrame and displays them.
python

Example:

```
import pandas as pd
```

```
df = pd.read_csv('data.csv')
columns_to_select = ['col1', 'col2', 'col3']
selected_columns = df[columns_to_select]
print(selected_columns)
```

Filtering rows from a DataFrame: Using pandas, write a script that filters rows from a DataFrame based on a specific condition and displays the resulting DataFrame.

Example:

```
import pandas as pd

df = pd.read_csv('data.csv')
condition = df['col1'] > 5
filtered_df = df[condition]
print(filtered_df)
```

Calculating statistics on a NumPy array: Using NumPy , write a script that loads a CSV file into a NumPy array, and calculates the mean, standard deviation, and maximum value of the array.

Example:

```
import numpy as np

data = np.genfromtxt('data.csv', delimiter=',')
mean = np.mean(data)
std = np.std(data)
max_value = np.max(data)
print("Mean:", mean)
print("Standard deviation:", std)
print("Max value:", max_value)
```

Matrix operations: Using NumPy, write a script that creates a matrix and performs basic mathematical operations such as addition, subtraction, multiplication and transpose of matrix.

```
import numpy as np

matrix1 = np.array([[1,2,3],[4,5,6]])
matrix2 = np.array([[7,8,9],[10,11,12]])

#Addition
add_matrix = matrix1 + matrix2
print("Addition of matrices:", add_matrix)

#Subtraction
```

```
sub_matrix = matrix1 - matrix2
print("Subtraction of matrices:", sub_matrix)

#multiplication
mul_matrix = matrix1 * matrix2
print("Multiplication of matrices:", mul_matrix)
```

Лекция 3

Работа с объектами MS Office

Модуль win32com на Python позволяет взаимодействовать с приложениями Microsoft Office, такими как Word, Excel и PowerPoint, с помощью интерфейса Component Object Model (COM).

Например, вы можете использовать его, чтобы открыть существующий документ Word, добавить текст, изменить шрифт и сохранить документ:

```
import win32com.client as win32

word = win32.gencache.EnsureDispatch('Word.Application')
doc = word.Documents.Open('C:\\path\\to\\document.docx')
word.Visible = True

range = doc.Range()
range.InsertAfter("Hello, world!")
range.Font.Name = "Arial"
range.Font.Size = 14

doc.Save()
doc.Close()
word.Quit()
```

В Excel вы можете открыть существующую книгу, добавить данные на лист, создать диаграммы и графики и сохранить книгу:

```
import win32com.client as win32

excel = win32.gencache.EnsureDispatch('Excel.Application')
workbook = excel.Workbooks.Open('C:\\path\\to\\workbook.xlsx')
excel.Visible = True

worksheet = workbook.Worksheets("Sheet1")
worksheet.Cells(1, 1).Value = "Name"
worksheet.Cells(1, 2).Value = "Age"
worksheet.Cells(2, 1).Value = "John"
worksheet.Cells(2, 2).Value = 30

chart = worksheet.ChartObjects().Add(0,0,300,300).Chart
chart.ChartType = win32.constants.xlColumnClustered
chart.SetSourceData(worksheet.Range("A1:B2"))

workbook.Save()
workbook.Close()
```

```
excel.Quit()
```

Вы также можете использовать win32com для взаимодействия с PowerPoint, например, вы можете создать новую презентацию, добавить слайды, добавить текст и изображения и сохранить презентацию:

```
import win32com.client as win32

ppt = win32.gencache.EnsureDispatch('PowerPoint.Application')
presentation = ppt.Presentations.Add()
ppt.Visible = True

slide = presentation.Slides.Add(1, win32.constants.ppLayoutTitleOnly)
title = slide.Shapes.Title
title.TextFrame.TextRange.Text = "My Presentation"

slide = presentation.Slides.Add(2, win32.constants.ppLayoutBlank)
textbox = slide.Shapes.AddTextbox(1,20,200,100,50)
textbox.TextFrame.TextRange.Text = "Hello, World!"

image = slide.Shapes.AddPicture("C:\\path\\to\\image.jpg", False, True, 20, 20, 100, 100)

presentation.SaveAs("C:\\path\\to\\presentation.pptx")
presentation.Close()
ppt.Quit()
```

Работать с файлами .docx и .xlsx в Python можно с помощью библиотек docx и openpyxl соответственно.

docx позволяет читать и редактировать существующие файлы .docx, а также создавать новые файлы .docx с нуля. В файл .docx можно добавлять текст, абзацы, списки, таблицы и изображения.

Например, вы можете использовать следующий код, чтобы создать новый файл .docx и добавить к нему заголовок и некоторый текст:

```
from docx import Document

document = Document()
document.add_heading('My Heading', 0)
document.add_paragraph('My paragraph')
document.save('My document.docx')
```

openpyxl позволяет читать и записывать файлы .xlsx. Вы можете получить доступ к ячейкам, строкам и столбцам электронной таблицы и изменить их значения. Вы также

можете создавать новые электронные таблицы и добавлять в них диаграммы, формулы и изображения.

Например, вы можете использовать следующий код для чтения данных из существующего файла .xlsx и печати значения ячейки в первой строке и первом столбце:
from openpyxl import load_workbook

```
workbook = load_workbook('data.xlsx')
worksheet = workbook.active
cell_value = worksheet.cell(row=1, column=1).value
print(cell_value)
```

Объектно-ориентированное программирование (ООП) — это парадигма программирования, основанная на концепции объектов, обладающих свойствами (также известными как атрибуты) и методами. В Python классы используются для определения объектов, их свойств и методов.

Класс — это план создания объектов (определенной структуры данных), предоставления начальных значений для состояния (переменные-члены или атрибуты) и реализации поведения (функции-члены или методы).

Вот пример простого класса в Python:

```
class Dog:
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed

    def bark(self):
        print("Woof!")

dog1 = Dog("Fido", "Golden Retriever")
dog1.bark() # Output: Woof!
```

В этом примере мы определяем класс Dogc двумя атрибутами nameи breed. Метод __init__— это специальный метод, который вызывается при создании нового объекта этого класса. Он инициализирует атрибуты объекта. В классе также есть метод bark, который печатает "Гав!" когда звонили.

Затем мы можем создать объект класса Dog, вызвав имя класса как функцию и передав необходимые аргументы для __init__метода.

В ООП классы наследуются от других классов и могут добавлять или переопределять свои методы и атрибуты. Это обеспечивает чистую и организованную структуру кода с

меньшим количеством повторяющихся кодов. Вы также можете использовать наследование для создания иерархии классов, которая организует классы в соответствии с их функциональностью и отношениями.

Пример класса для банковского счета с методами пополнения, снятия и проверки баланса:

```
class BankAccount:
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"{amount} has been deposited to {self.name}'s account")

    def withdraw(self, amount):
        if amount > self.balance:
            print(f"Insufficient balance in {self.name}'s account")
        else:
            self.balance -= amount
            print(f"{amount} has been withdrawn from {self.name}'s account")

    def check_balance(self):
        print(f"{self.name}'s current balance is {self.balance}")

account1 = BankAccount("John Doe", 1000)
account1.check_balance() # Output: John Doe's current balance is 1000
account1.deposit(500) # Output: 500 has been deposited to John Doe's account
account1.check_balance() # Output: John Doe's current balance is 1500
account1.withdraw(2000) # Output: Insufficient balance in John Doe's account
account1.withdraw(500) # Output: 500 has been withdrawn from John Doe's account
account1.check_balance() # Output: John Doe's current balance is 1000
```

Этот класс имеет три метода: deposit, withdraw, и check_balance. Каждый метод выполняет определенное действие над объектом банковского счета. Метод depositпринимает сумму в качестве входных данных и добавляет ее к балансу счета. Метод withdraw принимает сумму в качестве входных данных, проверяет, есть ли на счете достаточный баланс, и если да, то вычитает сумму из баланса. Метод check_balance просто печатает текущий баланс счета.

В этом примере мы создаем объект account1класса BankAccountи вызываем методы объекта, чтобы продемонстрировать, как работает класс.

Другой пример — создание класса для прямоугольника с атрибутами длины и ширины и методами вычисления площади и периметра:

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

    def calculate_perimeter(self):
        return 2 * (self.length + self.width)

rect1 = Rectangle(5, 10)
print(rect1.calculate_area()) # Output: 50
print(rect1.calculate_perimeter()) # Output: 30
```

В этом примере класс Rectangle имеет два атрибута: lengthи width. Он также имеет два метода, calculate_area и calculate_perimeter, которые возвращают площадь и периметр прямоугольника соответственно.

Вы также можете создать класс для университета с атрибутами для имени, адреса и студентов, а также методами добавления и удаления студентов и расчета количества студентов.

Вы можете использовать свое воображение и придумать различные сценарии и приложения, которые могли бы использовать классы и объекты.

Практические задачи по ООП:

- Создайте класс для банковского счета с методами пополнения, снятия и проверки баланса.
- Создайте класс для автомобиля с методами запуска, остановки и подачи сигнала.
- Создайте класс для человека с атрибутами для имени, возраста и адреса.
- Создайте класс для университета с атрибутами имени, адреса и студентов.
- Создайте класс для прямоугольника с атрибутами длины и ширины и методами вычисления площади и периметра.
- Создайте класс для корзины покупок с методами добавления и удаления товаров и расчета общей стоимости.
- Создайте класс для игры с атрибутами для очков, жизней и уровня, а также методы для запуска, завершения и перезапуска игры.

Дополнительную информацию и примеры вы можете найти на официальном сайте python: https://docs.python.org/3/tutorial/classes.html.

Лекция 4

Оптимизация, линейная алгебра и машинное обучение

Оптимизация — это процесс поиска наилучшего решения проблемы в рамках набора ограничений. С математической точки зрения, это включает в себя поиск значений набора переменных, которые минимизируют или максимизируют заданную функцию, называемую целевой функцией.

Оптимизация широко используется во многих областях, таких как финансы, инженерия и исследование операций. Существует множество различных методов оптимизации, таких как линейное программирование, нелинейное программирование, целочисленное программирование и стохастическая оптимизация, которые можно использовать для решения различных типов задач оптимизации.

В Python есть несколько библиотек, таких как scipy.optimize, которые предоставляют широкий спектр алгоритмов оптимизации, таких как градиентный спуск, Nelder-Mead, сопряженный градиент и BFGS. Эти библиотеки можно использовать для оптимизации различных типов функций, включая линейные и нелинейные функции, с ограничениями или без них.

SciPy — это библиотека, содержащая набор научных и численных алгоритмов, включая оптимизацию, линейную алгебру, интеграцию, интерполяцию и многое другое. Он построен на основе NumPy и предоставляет дополнительные функции для научных вычислений.

Например, вы можете использовать функцию минимизации из модуля scipy.optimize, чтобы минимизировать функцию.
Вы можете использовать следующий код для минимизации функции f(x) = x^2 с помощью алгоритма BFGS:

```
from scipy.optimize import minimize

def f(x):
    return x**2

res = minimize(f, x0=2, method='BFGS')
print(res.x) #The minimum point
```

Также можно оптимизировать функции с ограничениями, используя функцию минимизации, предоставив словарь ограничений или используя метод «SLSQP»:

```
from scipy.optimize import minimize

def f(x):
    return x[0]**2 + x[1]**2
```

```
def constraint1(x):
    return x[0] + x[1] - 1

def constraint2(x):
    return 1 - x[0] - x[1]

cons = [{'type': 'eq', 'fun': constraint1},
        {'type': 'ineq', 'fun': constraint2}]

res = minimize(f, x0=[0.5, 0.5], constraints=cons)
print(res.x) #The minimum point
```

Линейная алгебра — это раздел математики, который занимается векторными пространствами и линейными преобразованиями между ними. Это фундаментальный инструмент во многих областях математики и естественных наук, включая физику, инженерию, информатику и экономику.

В Python самой популярной библиотекой для линейной алгебры является NumPy , которая предоставляет широкий набор функций и классов для работы с массивами и матрицами. Это также основа для других важных библиотек, таких как scikit-learn и pandas.

NumPy предоставляет функции для базовых операций линейной алгебры, таких как умножение матриц, инверсия и вычисление определителя, а также для более сложных операций, таких как разложение собственных и сингулярных значений.

Например, вы можете использовать функцию точки для выполнения матричного умножения:

```
import numpy as np

A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.dot(A, B)
print(C)
```

Вывод будет:

```
[[19 22]
 [43 50]]
```

Вы также можете использовать функцию inv для вычисления обратной матрицы:

```
import numpy as np
```

```
A = np.array([[1, 2], [3, 4]])
A_inv = np.linalg.inv(A)
print(A_inv)
```

Вывод будет:
```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Кроме того, модуль NumPy .linalg также предоставляет функции для решения линейных систем уравнений, вычисления собственных значений и собственных векторов и разложения по сингулярным числам.

Линейная алгебра является фундаментальной концепцией в машинном обучении и науке о данных. Она используется во многих алгоритмах, таких как анализ главных компонентов (PCA), разложение по сингулярным значениям (SVD), линейный дискриминантный анализ (LDA) и многие другие.

Scikit-learn, библиотека машинного обучения, построенная на основе NumPy , использует возможности линейной алгебры NumPy и предоставляет широкий спектр моделей машинного обучения, которые можно использовать для таких задач, как классификация, регрессия и кластеризация.

Машинное обучение — это область искусственного интеллекта (ИИ), которая включает разработку алгоритмов и статистических моделей, которые позволяют компьютерам учиться и делать прогнозы или решения без явного программирования для этого.

Существует несколько видов машинного обучения:

- Контролируемое обучение: алгоритм обучается на размеченном наборе данных, где для каждого входа предоставляется правильный результат. Цель состоит в том, чтобы делать прогнозы на основе новых, невидимых данных. Примеры включают линейную регрессию, логистическую регрессию и машины опорных векторов.

- Неконтролируемое обучение: алгоритм обучается на немаркированном наборе данных, и цель состоит в том, чтобы обнаружить закономерности или структуру в данных. Примеры включают кластеризацию k-средних и анализ основных компонентов (PCA).

- Обучение с подкреплением: алгоритм учится, взаимодействуя с окружающей средой, получая награды или штрафы за определенные действия. Цель состоит в том, чтобы научиться принимать решения, которые со временем максимизируют совокупное вознаграждение.

- Глубокое обучение: Подобласть машинного обучения, которая включает обучение глубоких нейронных сетей (ГНС) на больших объемах данных. Было показано, что DNN достигают самых современных результатов в широком спектре задач, таких как классификация изображений, обработка естественного языка и распознавание речи.

Scikit-learn — это широко используемая библиотека для машинного обучения на Python. Она предоставляет широкий спектр инструментов и функций для контролируемого и неконтролируемого обучения, включая поддержку сред глубокого обучения, таких как TensorFlow и PyTorch.

Например, вы можете использовать scikit-learn для обучения модели линейной регрессии на наборе данных:

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston

# Load the Boston housing dataset
boston = load_boston()
X = boston.data
y = boston.target

# Create and fit the model
model = LinearRegression()
model.fit(X, y)

# Make predictions
predictions = model.predict(X)
```

В этом примере класс LinearRegression используется для создания модели линейной регрессии, метод fit используется для обучения модели на наборе данных о жилье в Бостоне, а метод predict используется для прогнозирования новых данных.

Машинное обучение широко используется в NLP для таких задач, как:

- Классификация текста: классификация текста по предопределенным категориям, таким как обнаружение спама, анализ настроений и классификация тем.
- Распознавание именованных сущностей (NER): идентификация и извлечение сущностей, таких как люди, организации и местоположения, из текста.
- Тегирование части речи (POS): маркировка слов в предложении их грамматическими ролями, таких как существительные, глаголы и прилагательные.
- Моделирование языка: предсказание следующего слова в предложении по предыдущим словам.
- Машинный перевод: перевод текста с одного языка на другой.

- Обобщение текста: автоматическое обобщение основных моментов текста.
- Анализ тональности: определение тональности заданного текста.

Например, вы можете использовать библиотеку scikit-learn для обучения модели классификации текста в наборе данных:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

#define dataset
X_train = ["This is a positive text", "This is a negative text", "This is a neutral text"]
y_train = ["positive", "negative", "neutral"]
X_test = ["This is a positive text", "This is a negative text"]

#vectorize text
vectorizer = CountVectorizer()
X_train_vectors = vectorizer.fit_transform(X_train)
X_test_vectors = vectorizer.transform(X_test)

#train model
clf = MultinomialNB()
clf.fit(X_train_vectors, y_train)

#predict
predicted = clf.predict(X_test_vectors)
print("Accuracy: ", accuracy_score(predicted, y_test))
```

В этом примере набор данных представляет собой список текстов и соответствующих им меток («положительно», «отрицательно», «нейтрально»). Класс CountVectorizer используется для преобразования текстов в векторы числовых признаков, класс MultinomialNB используется для обучения наивной байесовской модели на наборе данных, а метод predict используется для прогнозирования новых данных.

Ещё один пример, в котором вы можете использовать библиотеку scikit-learn: обучение модели распознавания именованных объектов (NER) в наборе данных:

```
import nltk
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction import DictVectorizer
from sklearn.tree import DecisionTreeClassifier
```

```
#define dataset
sentences = [["I", "live", "in", "New York"], ["I", "work", "at", "Google"]]
pos_tags = [["PRP", "VBP", "IN", "NNP"], ["PRP", "VB", "IN", "NNP"]]
ner_tags = [["O", "O", "O", "B-GPE"], ["O", "O", "O", "B-ORG"]]

#flatten the data
sentences = [token for sent in sentences for token in sent]
pos_tags = [tag for sent in pos_tags for tag in sent]
ner_tags = [tag for sent in ner_tags for tag in sent]

#extract features
def extract_features(sentence, index):
    features = {
        'word': sentence[index],
        'is_first': index == 0,
        'is_last': index == len(sentence) - 1,
        'is_capitalized': sentence[index][0].upper() == sentence[index][0],
        'is_all_caps': sentence[index].upper() == sentence[index],
        'is_all_lower': sentence[index].lower() == sentence[index],
        'prefix-1': sentence[index][0],
        'prefix-2': sentence[index][:2],
        'prefix-3': sentence[index][:3],
        'suffix-1': sentence[index][-1],
        'suffix-2': sentence[index][-2:],
        'suffix-3': sentence[index][-3:],
        'prev_word': '' if index == 0 else sentence[index - 1],
        'next_word': '' if index == len(sentence) - 1 else sentence[index + 1],
        'has_hyphen': '-' in sentence[index],
        'is_numeric': sentence[index].isdigit(),
        'capitals_inside': sentence[index][1:].lower() != sentence[index][1:]
    }
    return features

#extract features for each token
X = [extract_features(sentences, i) for i in range(len(sentences))]

#Vectorize the features
vec = DictVectorizer()
X = vec.fit_transform(X)

#split dataset into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, ner_tags, test_size=0.33)

#train the model
```

```
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

#predict
predicted = clf.predict(X_test)
print("Accuracy: ", accuracy_score(predicted, y)
```

Помимо классификации текста, еще одна распространенная задача в NLP, к которой можно подойти с помощью машинного обучения, называется распознаванием сущностей (NER). Это включает в себя идентификацию и классификацию именованных объектов, таких как люди, организации и местоположения, в тексте. Один из подходов к NER заключается в использовании алгоритма маркировки последовательностей, такого как скрытая марковская модель (HMM) или условное случайное поле (CRF), для прогнозирования меток для каждого слова в предложении. Другой подход заключается в использовании модели машинного обучения, такой как машина опорных векторов (SVM) или нейронная сеть, для классификации слов или фраз в тексте.

На практике многие задачи NLP, такие как определение частей речи и синтаксический анализ, также могут решаться с помощью машинного обучения. Еще одним примером задачи НЛП, к которой можно подойти с помощью машинного обучения, является анализ настроений. Анализ тональности — это процесс определения того, выражает ли фрагмент текста положительное, отрицательное или нейтральное настроение. Анализ настроений используется в широком спектре приложений, таких как обслуживание клиентов, маркетинг и анализ общественного мнения. Общие методы анализа настроений включают использование контролируемых алгоритмов машинного обучения для классификации текста на основе наличия определенных ключевых слов или использование неконтролируемых методов для выявления шаблонов в тексте.

Лекция 5

Обрабюотка изображений

Обработка изображений — это метод обработки и анализа цифровых изображений, который часто используется в компьютерном зрении и компьютерной графике. OpenCV (Библиотека компьютерного зрения с открытым исходным кодом) — популярная библиотека с открытым исходным кодом для обработки изображений и компьютерного зрения на Python. Он предоставляет широкий спектр инструментов и алгоритмов для обработки изображений, таких как фильтрация изображений, пороговое значение, преобразование цветового пространства, обнаружение признаков и многое другое.

Одной из наиболее распространенных задач при обработке изображений является определение порога изображения, то есть процесс преобразования изображения в бинарное изображение (черно-белое) на основе порогового значения. OpenCV предоставляет несколько методов определения порога, таких как бинарное определение порога, адаптивное определение порога и определение порога Оцу.

Другой распространенной задачей обработки изображений является фильтрация изображений, то есть процесс изменения пикселей изображения на основе ядра фильтра. OpenCV предоставляет несколько методов фильтрации изображений, таких как размытие по Гауссу, срединное размытие и двусторонний фильтр.

Ещё одной распространенной задачей обработки изображений является обнаружение признаков, т. е. процесс обнаружения и извлечения элементов изображения, таких как углы, края и круги. OpenCV предоставляет несколько методов обнаружения функций, таких как обнаружение углов Харриса, обнаружение углов FAST и преобразование круга Хафа.

OpenCV может быть использовано для обнаружения лиц на изображении. Это можно сделать с помощью предварительно обученного классификатора, такого как алгоритм Виолы-Джонса, который включен в OpenCV. Другой задачей может быть использование OpenCV для обнаружения и отслеживания объектов в видеопотоке, что может быть выполнено с использованием различных алгоритмов, таких как вычитание фона, оптический поток и отслеживание объектов.

Примеры использования OpenCV для задач обработки изображений.

Пороговое значение: следующий код демонстрирует, как использовать OpenCV для выполнения двоичного порогового значения для изображения. Функция

cv2.threshold() используется для преобразования изображения в двоичное изображение на основе порогового значения.

Пример

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Perform binary thresholding
ret, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Show the original and thresholded images
cv2.imshow("Original", img)
cv2.imshow("Thresholded", thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Фильтрация: следующий код демонстрирует, как использовать OpenCV для выполнения медианной фильтрации изображения. Функция cv2.medianBlur()используется для применения медианного фильтра к изображению.

Пример

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Perform median filtering
median = cv2.medianBlur(img, 5)

# Show the original and filtered images
cv2.imshow("Original", img)
cv2.imshow("Median Filtered", median)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Обнаружение функций. В следующем коде показано, как использовать OpenCV для обнаружения и рисования кругов на изображении. Эта cv2.HoughCircles()функция используется для обнаружения кругов на изображении с помощью преобразования круга Хафа.

Пример

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect circles in the image
circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1, 20, param1=50, param2=30, minRadius=0, maxRadius=0)

# Draw the circles on the image
for circle in circles[0, :]:
    cv2.circle(img, (circle[0], circle[1]), circle[2], (0, 255, 0), 2)

# Show the original and processed images
cv2.imshow("Original", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Бинаризация — это процесс преобразования изображения в бинарное изображение, где пиксели либо черные, либо белые. Порог для бинаризации обычно устанавливается на основе интенсивности или цвета изображения. Например, в изображениях в градациях серого пиксели со значениями интенсивности выше определенного порога устанавливаются белыми, а пиксели со значениями интенсивности ниже порога — черными. В OpenCV доступно несколько методов бинаризации, таких как бинаризация Оцу, адаптивная бинаризация и бинаризация Ниблэка.

Дилатация и эрозия — это операции морфологической обработки изображения, которые используются для расширения или сжатия границ объекта на изображении. Расширение увеличивает размер объекта, добавляя пиксели к

границе объекта, а эрозия уменьшает размер объекта, удаляя пиксели с границы объекта. Эти операции полезны для удаления шума и небольших отверстий в изображении.

Методы поиска контуров на изображении включают использование алгоритмов обнаружения краев, таких как детектор краев Кэнни, или использование методов пороговой обработки. Контуры можно использовать для идентификации и сегментации объектов на изображении.

Обнаружение и сегментация объектов на изображении может быть достигнуто с помощью нескольких методов, таких как определение порога, обнаружение краев и обнаружение контуров. Эти методы можно использовать для идентификации и выделения определенных объектов на изображении, например объектов определенного типа или определенного цвета.

Например, используя OpenCV, мы можем загрузить изображение, а затем использовать детектор краев Канни, чтобы найти края на изображении. Затем мы можем использовать метод findContours, чтобы найти контуры на изображении. Затем мы можем нарисовать контуры на изображении, чтобы сегментировать объекты на изображении.

Пример

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Use Canny edge detector to find edges in the image
edges = cv2.Canny(gray, 50, 150)

# Find contours in the image
contours, hierarchy = cv2.findContours(edges, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# Draw the contours on the image
cv2.drawContours(img, contours, -1, (0, 255, 0), 2)
```

```
# Show the image
cv2.imshow("Segmented Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вы также можете использовать inRangeметод для порогового изображения и сегментации объекта определенного цвета.

Пример

```
import cv2

# Load the image
img = cv2.imread("image.jpg")

# Define the range of the color you want to segment
lower = (0, 0, 0)
upper = (50, 50, 50)

# Threshold the image to get only the desired color
mask = cv2.inRange(img, lower, upper)

# Show the segmented image
cv2.imshow("Segmented Image", mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
```